Cancel

```
#!/usr/bin/perl
use strict;
#Correl_Display_1_6_1.pl
#Designed to take the CVS formatted exported file from OmniViz and produce a nice PNG
#image similar to that on the screen in OmniViz
#New in Version 1.1:
#       Inclusion of clinical data!;
use GD;
$|=1;                                   #Do not use output buffer - print diag immediately
########################
#Global Variable decision area:
my %Config;                             #Main Configuration hash.
my $Top_Color=0;
#my $Block_Size                         = 10;  #The size (in Pixels) of each block.
#File names: Hard Wired in version 1_1!
#my $Clinical_Data_File                 = "./Klinisch_data_AML.csv";  #The name of the Clinical
Datafile (Comma delimited format).
#my $Output_File                        = "Output.png";                    #Name of the
final generated image.

#Other parameters:
#my $Block_Lines                        = "F";  #Whether to draw lines round the (inside) of
the blocks
                                        #NB: Reduces colored area by 1 pixel in both
dimensions
#my $Draw_Key_F                         = "T";  #Should a Key be prepared?
#my $Color_Strips               = 40;   #The number of intervening colors in the 'Strip'
#my $Minimum                    = -1;   #Assumed minmum of correlation data
#my $Maximum                    = +1;   #Assumed minmum of correlation data
#my $Scale                      = 5;    #The multiplication factor for relative to $Block_Size
of the Blocks in the Color Stripe

########################
Load_Configuration ();                  #Load configuration from STDIN

########################File acceptance testing########################
$Config{Correlation_File}     = shift @ARGV;            #Pull filename from ARGV
$Config{Output_File}   = shift @ARGV;
if (($Config{Correlation_File} eq "") or !(-e $Config{Correlation_File})) #Check file
exists (and is not blank!)
        {die "Please enter valid Correlation file name: \n'",$Config{Correlation_File},"'
Appears to be invalid\n";}
if ($Config{Output_File} eq "")
        {warn "Output filename not specified: defaulting to 'Output.png' (all previous files
of same name will be over written) Hit !!!Ctrl-C!!! NOW to avoid\n";}



open IP_FILE, $Config{Correlation_File} or          #Open input file or exit with error
        die "Cannot open '",$Config{Correlation_File},"'\n for some reason\n";

########################Declare useful variables########################
my @IDs;                #Global - for when we find them.
my $Row=0;              #Need this for later when loading data.
my $Max_Col=-1;                 #Used more as a security check than actually in processing.
my @Matrix;             #Main Matrix loaded.
my %Patient_ID;                 #Hash array to store the patient IDs: Used to linke the CC &
Clinical data
########################Load data from Correlation Matrix file########################
while (<IP_FILE>)
        {
        chomp ();                       #Remove end of line char
        $_ =~ s/[\n\r]//g;
        if ($_ eq "") {next;}           #In case there are any blank lines
        unless (/\,/) {die "Errr.  There is a distinct lack of commas on this line...of the
Correlation_File: '",$Config{Correlation_File},"':\n'",substr ($_,0,20) ,"....'\n";}
        my @Fields = split (",",$_);  #Split on Commas (it is a Comma delimited file);
        if (/^Variables/)               #Ie. The first line with the "names" of the
rows/colums.
                {
                shift @Fields; #Strip the 'Variables' part off.
```

Figure 15a

Cancel

```
#               print "@Fields\n";
                @IDs = @Fields;         #Take of copy of the '@Fields' Array which is Locally
scoped
                next;  #Skip to next line
                }
        my $Patient_ID = shift @Fields; #Strip the 'Patient' part off the front of each
line.
#       print "D: Loading CC data for patient ID: '$Patient_ID'\n";
        $Patient_ID{$Row} = $Patient_ID;
        if ($Patient_ID         =~ m/b$/)
                {
                print "D: Detected 'b' suffix Patient: '$Patient_ID' Corrected to:";
                $Patient_ID =~ s/b$//;
                print " '$Patient_ID'\n";
                }
        if ($#Fields != $Max_Col)       #Check consistent number of Coloums reported
                {
                if ($Max_Col == -1)
                        {
                        $Max_Col = $#Fields;  #Wasteful to do this every time..
                        print "D: Setting Max_Col to: '$Max_Col'\n";
                        }
                else
                        {
                        print "D: Warning: Number of Coloums Deviation: Row '$Row' (has
'$#Fields' coloums, previous ones had '$Max_Col'\n";
                        }
                }


        foreach my $C_Col (0..$#Fields)
                {
                $Matrix[$Row][$C_Col] = $Fields[$C_Col];
                }
        $Row++;
        }

print "D: Matrix is: [Rows x Coloums]: $Row x $Max_Col\n";
print "D: Or to put it another way: ",$#Matrix, " x ",$#{$Matrix[0]},"\n";
print "D: Matrix Test cell = 0,0 = $Matrix[0][0]\n D: Matrix Test cell 1,0 = $Matrix[1][0]
D: Matrix Test cell 303,303 = $Matrix[302][302]\n";
print "D: We are using clinical data file: '$Config{Clinical_Data_File}'\n";
open CLIN_FILE, $Config{Clinical_Data_File} or
        die "Cannot open clinical datafile: '",$Config{Clinical_Data_File},"' for some
reason\n";
my $Clinical_Data_Col_Header_Text_1;
my $Clinical_Data_Col_Header_Text_2;
my $Clinical_Data_Col_Header_Text_3;
my $Clinical_Data_Col_Header_Text_4;
my $Clinical_Data_Col_Header_Text_5;
my $Clinical_Data_Col_Header_Text_6;
my $Clinical_Data_Col_Header_Text_7;
my $Clinical_Data_Col_Header_Text_8;
my $Clinical_Data_Col_Header_Text_9;
my $Wanted_Header_Col_Index_1;
my $Wanted_Header_Col_Index_2;
my $Wanted_Header_Col_Index_3;
my $Wanted_Header_Col_Index_4;
my $Wanted_Header_Col_Index_5;
my $Wanted_Header_Col_Index_6;
my $Wanted_Header_Col_Index_7;
my $Wanted_Header_Col_Index_8;
my $Wanted_Header_Col_Index_9;
my %Classification_1;
my %Classification_2;
my %Classification_3;
my %Classification_4;
my %Classification_5;
my %Classification_6;
my %Classification_7;
my %Classification_8;
```

Figure 15b

Cancel

```
my %Classification_9;
while (<CLIN_FILE>)
        {
        chomp ();       #Death to New Line characters! (;-)
        unless (/\,/) {die "Errr.  There is a distinct lack of commas on this line...of the
Correlation_File: '",$Config{Correlation_File},"':\n'",substr ($_,0,20),"....'\n";}
        my @Fields = split (",",$_);
        if (/^Volgnummer/)     #Match the Header line:
                {
                print "D:'$_'\n";
#               @Clinical_Data_Col_Headers = @Fields;       #i.e. just copy the comma-split
line
#Run through all column headers to find the index of the one we are looking for:
                foreach my $C_Column (0..scalar (@Fields))
                        {
                        if ($Fields[$C_Column] eq $Config{Header_Col_1})     #Scan across the
header line for column we want #1
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_1 = $C_Column;
                                $Clinical_Data_Col_Header_Text_1 = $Config{Header_Col_1};
        #Only now will we add it.
                                print "D: Found the Coloumn [1] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_1'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_2})     #Scan across the
header line for column we want #2
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_2 = $C_Column;
                                $Clinical_Data_Col_Header_Text_2 = $Config{Header_Col_2};
        #Only now will we add it.
                                print "D: Found the Coloumn [2] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_2'\n";
                                $Clinical_Data_Col_Header_Text_2 =~ s/,/\./g;       #Sometimes
being Dutch is cute, othertimes its just plain annoying...Ja?
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_3})     #Scan across the
header line for column we want #1
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_3 = $C_Column;
                                $Clinical_Data_Col_Header_Text_3 = $Config{Header_Col_3};
        #Only now will we add it.
                                print "D: Found the Coloumn [3] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_3'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_4})     #Scan across the
header line for column we want #1
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_4 = $C_Column;
                                $Clinical_Data_Col_Header_Text_4 = $Config{Header_Col_4};
        #Only now will we add it.
                                print "D: Found the Coloumn [4] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_4'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_5})     #Scan across the
header line for column we want #1
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_5 = $C_Column;
                                $Clinical_Data_Col_Header_Text_5 = $Config{Header_Col_5};
        #Only now will we add it.
                                print "D: Found the Coloumn [5] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_5'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_6})     #Scan across the
header line for column we want #1
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_6 = $C_Column;
```

Figure 15c

```
                                $Clinical_Data_Col_Header_Text_6 = $Config{Header_Col_6};
        #Only now will we add it.
                                print "D: Found the Coloumn [6] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_6'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_7})    #Scan across the
header line for column we want #7
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_7 = $C_Column;
                                $Clinical_Data_Col_Header_Text_7 = $Config{Header_Col_7};
        #Only now will we add it.
                                print "D: Found the Coloumn [7] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_7'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_8})    #Scan across the
header line for column we want #7
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_8 = $C_Column;
                                $Clinical_Data_Col_Header_Text_8 = $Config{Header_Col_8};
        #Only now will we add it.
                                print "D: Found the Coloumn [8] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_8'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }
                        if ($Fields[$C_Column] eq $Config{Header_Col_9})    #Scan across the
header line for column we want #7
                                {       #Whoppie!  Found the one we want!
                                $Wanted_Header_Col_Index_9 = $C_Column;
                                $Clinical_Data_Col_Header_Text_9 = $Config{Header_Col_9};
        #Only now will we add it.
                                print "D: Found the Coloumn [9] in the header we are looking
for!: Index is: '$Wanted_Header_Col_Index_9'\n";
                                next;   #There is (we assume) only one unique coloumn name...
                                }



                        }
                if ($Clinical_Data_Col_Header_Text_1 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_1},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}
                if ($Clinical_Data_Col_Header_Text_2 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_2},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}
                if ($Clinical_Data_Col_Header_Text_3 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_3},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}
                if ($Clinical_Data_Col_Header_Text_5 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_5},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}

                if ($Clinical_Data_Col_Header_Text_7 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_7},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}

                if ($Clinical_Data_Col_Header_Text_8 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_8},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}

                if ($Clinical_Data_Col_Header_Text_9 eq "") #I.e., nothing was set...
                        {die "Opps.\nI was looking for the column header:
'",$Config{Header_Col_9},"' in the clinical data file: '",$Config{Clinical_Data_File},"'\nI
didn't find it!\nWhat I did find was: '",join (";",@Fields),"' if that helps...\n";}
```

Canc

Figure 15d

```
                next;           #We have found the Coloumn that we are looking for...so skip
to next line.
                }
#       print "D: Loading Clinical Classification for Patient: '$Fields[0]'  this
is:'$Fields[$Wanted_Header_Col_Index_1]' & :'$Fields[$Wanted_Header_Col_Index_2]' &:
'$Fields[$Wanted_Header_Col_Index_3]' &: '$Fields[$Wanted_Header_Col_Index_4]' &:
'$Fields[$Wanted_Header_Col_Index_5]'\n";           #The first field contains the header
Patient ID...
#       if (exists $Classification{$Fields[$Wanted_Header_Col_Index]})
#               {#We already have one of these!
#               die "Error!  Patient IDs are not unique!\nThis one
'",$Classification{$Fields[$Wanted_Header_Col_Index]},"' found for the 2nd time!";
        }
        $Classification_1{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_1];
        $Classification_2{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_2];
        $Classification_3{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_3];
        $Classification_4{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_4];
        $Classification_5{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_5];
        $Classification_6{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_6];
        $Classification_7{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_7];
        $Classification_8{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_8];
        $Classification_9{$Fields[0]} = $Fields[$Wanted_Header_Col_Index_9];
#       push @Classification, $Fields[$Wanted_Header_Col_Index];   #We know which column we
want: so just add this one...
        }
########################Prepare colors########################
#$Image -> filledRectangle ($x1, $y1, $x2+20*$Catergory*$Config{Block_Size} , $y2,
$Block_color);
#This last expression is so that all the bars will fit on!  The 800 is a guess!
my $Width = $Config{Block_Size} * $Row + ($Config{Block_Size} + $Config{Graph_Space} * 8);
my $Height = $Config{Block_Size} * $Max_Col;
#Create Image canvases & Allocate basic colors to them:

my $Image = new GD::Image ($Width , $Height);               #Create main image 'Canvas'
my $White  = $Image -> colorAllocate (255,255,255); #Set first color (also background
color!)
Top_Color_Print();
#my $Blue = $Image -> colorAllocate (0,0,255);              #Allocate color 'Blue';
#my $Red = $Image -> colorAllocate (255,0,0);               #Allocate color 'Red';
my $Black= $Image -> colorAllocate (0,0,0);         #Allocate color 'Black';
Top_Color_Print();
my $Col_Stripe_Width = $Config{Block_Size} * $Config{Scale}  * ($Config{Color_Strips}+1);
my $Col_Stripe_Height = $Config{Block_Size} * $Config{Scale} ;
print "D: Color Stripe will be ($Col_Stripe_Width x $Col_Stripe_Height)\n";
my $Color_Stripe_IMG = new GD::Image ($Col_Stripe_Width, $Col_Stripe_Height);
$Color_Stripe_IMG -> colorAllocate (255,0,255);     #Set first color (also background
color!)


my $Title_Bar = new GD::Image ($Width , 100);
$Title_Bar -> colorAllocate (255,255,255);  #Set first color (also background color!)
#my $Blue = $Image -> colorAllocate (0,0,255);              #Allocate color 'Blue';
#my $Red = $Image -> colorAllocate (255,0,0);               #Allocate color 'Red';
$Title_Bar -> colorAllocate (0,0,0);        #Allocate color  'Black';

my $Patient_IDs = new GD::Image (400, $Height);
$Patient_IDs -> colorAllocate (255,255,255);#Set first color (also background color!)
#my $Blue = $Image -> colorAllocate (0,0,255);              #Allocate color 'Blue';
#my $Red = $Image -> colorAllocate (255,0,0);               #Allocate color 'Red';
$Patient_IDs -> colorAllocate (0,0,0);              #Allocate color 'Black';

#my $Image = new GD::Image (1000,100);      #HW: For testing Color Stripe...
my @Color_Stripe;
#Colors run: Full Blue - Partial Blues - Full White - Partial Reds - Full Red
print "D: Allocate 'Blues': \n";
foreach my $C_Color (0..($Config{Color_Strips}/2-1))        #Run: Full Blue to one level
below white
        {
        printf ("%3i ",$C_Color);
```

Canc

Figure 15e

```
        my $Blue_level = 255/($Config{Color_Strips}/2)*$C_Color;   #The (complex)
calculation for the color level
        print "D: Allocating Color: Blue_level = '$Blue_level'\n";        #works for the
red as well but without the "255-" part
        push @Color_Stripe, $Image -> colorAllocate ($Blue_level,$Blue_level,255);
#       $Color_Stripe_IMG -> colorAllocate (255,$Blue_level,$Blue_level);

        Top_Color_Print();
        }
#print "D: $#Color_Stripe, @Color_Stripe\n";                                   #Note down
the index of the color just allocated in a 'look-up' table
#print "D: Allocating White: < As mid point >";
push @Color_Stripe, $Image -> colorAllocate (255,255,255); #The 'White' is fixed.
#$Color_Stripe_IMG -> colorAllocate (255,255,255);
#Top_Color_Print();
#print "D: $#Color_Stripe, @Color_Stripe\n";
print "\nD: Allocate 'Reds': \n";
foreach my $C_Color (1..($Config{Color_Strips}/2))  #Run: one above 'white' to full red
        {
        printf ("%3i ",$C_Color);
        my $Red_level = 255 - 255/($Config{Color_Strips}/2)*$C_Color;
        print "D: Red_level = '$Red_level'\n";
        push @Color_Stripe, $Image -> colorAllocate (255,$Red_level,$Red_level);
#       $Color_Stripe_IMG -> colorAllocate (255,$Red_level,$Red_level);
#       Top_Color_Print();
        }
print "\n";
#print "D: $#Color_Stripe, @Color_Stripe\n";
print "D: Strip Colors = '@Color_Stripe'\n";

########################Build array image####################
#Build array
my $Range=sqrt ( ($Config{Maximum} - $Config{Minimum}) ** 2);      #Ok, so we know that for
Pearson CC it will be 2
my $BINS = $#Color_Stripe +1;
my $Bin_width= $Range / $BINS;
print "D: Possible BINS = '$BINS';   For Range = '$Range', so each bin is: '$Bin_width'
wide\n";
print "D: Building Array:\n";
print "D:    ";
foreach my $row (0..$#Matrix)                 #Cycle through all rows
        {
        foreach my $col (0..$Max_Col)         #Cycle through all coloumsn
                {
                if ($row == $col)      {last;}
                my ($x1,$x2,$y1,$y2,$color);         #Declare Intermediate variables
                my $value = $Matrix [$row][$col] - $Config{Minimum};        #Re-center the
data scale to +ve
#               print "D: value = '$value'   ";
#Calculate the color required using the same indices as lodged @Color_Stripe (NB:
Color_Stripe need not exist by this stage: OPTIMISES AWAY?)
                $color = int ($value / $Bin_width) +1 +1;         #The extra '+1' is becaa
#               print "\nD: Matrix Color = $color, \n";
#               $bin = int ($value / 1) * (1/ $Color_Strips +1);
#               print "D: Bin = ' $color '\n";
                if ( $color >= $BINS) {$color = $BINS;}
                $x1 = $Config{Block_Size} * $col; $x2 = $x1 + $Config{Block_Size}-1;
        #Top left to Bottom right of a square
                $y1 = $Config{Block_Size} * $row; $y2 = $y1 + $Config{Block_Size}-1;
#               die "HIT BLOCK";
#               print "D: x1 = $x1, x2 = $x2 ; y1 = $y1 ; y2 = $y2\n";
                if ($Patient_ID{$row} eq $Config{Marked_Patient})#        print "D: value =
'$value'\n";
                        {$color=$Black;}
                $Image -> filledRectangle ($x1,$y1,$x2, $y2, $color);      #Actually draw
the square at the correct location
#               $Image -> rectangle ($x1,$y1,$x2-1, $y2-1, $Black); #Outline the square
                }
printf ("%5i ",$row);          #Just a counter printed to the screen / stream.
#       die "HIT BLOCK\n";
        }
```

Cancel

Figure 15f

Cancel

```
print "\n";
if ($Config{Block_Lines} eq "T")      #Did the user request lines?
        {
        Draw_Lines_on_Image ();
        }

my $Classes; my $Class_Lowest_Color;

if ($Config{Mark_Patient_Data} eq "Y")
        {
        ($Class_Lowest_Color, $Classes) = Mark_Patient_Data ();
        }
print "D: Classes Returned = '$Classes'; number of colors needed:
'",$Class_Lowest_Color,"'\n";

#my $Classification_Stripe_IMG = new GD::Image ($Config{Block_Size} * $Classes *
$Config{Scale}, $Config{Block_Size} * $Config{Scale});
######################################## Invoke Draw_Key () if necessary
if ($Config {Draw_Color_Stripe} eq "T")
        {
        Draw_Color_Stripe ();
        }


#Combine the images and write them out:
my $Parent_Image = new GD::Image ($Width + 100, $Height + 200);          #Create final
image 'Canvas' into which others are merged
my $White      = $Parent_Image -> colorAllocate (255,255,255);      #Set first color (also
background color!)
my $Black      = $Parent_Image -> colorAllocate (0,0,0);            #Formally allocate color
'Black'
my $Patient_ID_Width = 250;
$Parent_Image -> copy ($Image, $Patient_ID_Width,100, 0, 0, $Width, $Height);     #Merge the
main heat-map / Patient Data.
$Parent_Image -> copy ($Patient_IDs, 0,100, 0,0, $Patient_ID_Width, $Height);     #Merge the
Patient IDs
$Parent_Image -> copy ($Color_Stripe_IMG,($Width - $Col_Stripe_Width)/2 +
$Patient_ID_Width, $Height + 100 + 100 - $Col_Stripe_Height, 0, 0, $Col_Stripe_Width,
$Col_Stripe_Height+1);
$Parent_Image -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        ($Width - $Col_Stripe_Width)/2 + $Patient_ID_Width - 40,
                        $Height + 100 + 40 + ($Config{Block_Size} * $Config{Scale}) /2,
                        "-1");

$Parent_Image -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $Width / 2 + 100 - 10,
                        $Height + 100 + 40 + ($Config{Block_Size} * $Config{Scale}) /2,
                        "0");

$Parent_Image -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        ($Width - $Col_Stripe_Width)/2 + $Patient_ID_Width +
$Col_Stripe_Width ,
                        $Height + 100 + 40 + ($Config{Block_Size} * $Config{Scale}) /2,
                        "+1");
my $x1=0;
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $x1, 90, "FAB");
$x1 = $x1 +$Config{Graph_Space};
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $x1, 90, "WBC");

$x1 = $x1 +$Config{Graph_Space};
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $x1, 90, "FLT3 ITD");

$x1 = $x1 +$Config{Graph_Space};
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $x1, 90, "OS");

$x1 = $x1 +$Config{Graph_Space};
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
```

Figure 15g

ANNOTATED SHEET

```
                        $x1, 90, "EFS");

$x1 = $x1 +$Config{Graph_Space};
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $x1, 90, "EVI1");

$x1 = $x1 +$Config{Graph_Space};
$Title_Bar -> stringTTF ($Black, "./fonts/arial.ttf", 30, 0,
                        $x1, 90, "CEBP mutant");


$Parent_Image -> copy ($Title_Bar, $Patient_ID_Width,      0,
                        0, 0, $Width, 100);
print "Just to remind you: the image created will be :'",$Config{Output_File},"' (you can
alter the default by using 2nd command line argument)\n";

$Parent_Image -> stringTTF ($Black, "./fonts/arial.ttf", 50, 3.142 / 2,
                        $Width  - 100,
                        $Height ,
                        "Orginal Correlation File: '$Config{Correlation_File}'");
$Parent_Image -> stringTTF ($Black, "./fonts/arial.ttf", 50, 3.142 / 2,
                        $Width  - 40,
                        $Height ,
                        "This Image is: '$Config{Output_File}'");


binmode OUTPUT;
open OUTPUT, ">$Config{Output_File}" or die "Cannot open output file: '" ,
$Config{Output_File},"'\n";
print OUTPUT $Parent_Image -> png ();              #Thankfully OO!  The difficult bit!
close OUTPUT;                           #Will close anyway upon program exit




#
#
#
#
#
#Subroutines only below here....
#
#       ##################################
#########SUB START
sub Draw_Lines_on_Image {

        print "D: Ok, You wanted lines....\n";       #Guess so....
        my $x_max = $Config{Block_Size} * $Max_Col; #Pre-calculate the right-hand edge
        my $y_max = $Config{Block_Size} * $Row;              #Pre-calculate the bottom edge.
        print "D: (Horizontal): ";
        foreach my $row (0..$Row)            #For all rows
                {
                my $y = $Config{Block_Size} * $row;  #Calculate the 'y' position
                $Image -> line (0, $y, $x_max, $y, $Black); #Draw Horizontal Line
#               printf ("%5i ",$row);
                }
print "\n";
        print "D: (Vertical): ";
        .foreach my $col (0..$Max_Col)        #For all coloumns
                {
                my $x = $Config{Block_Size} * $col;  #Calculate the 'x' position
                $Image -> line ($x, 0, $x, $y_max, $Black); #Draw Vertical Line
#               printf ("%5i ",$col);
                }
print "\n";
}


#########SUB START
sub Draw_Color_Stripe {
```

Figure 15h

Canc[illegible]

```
my $White  = $Color_Stripe_IMG -> colorAllocate (255,0,255);      #Set first color (also
background color!)
my $Black  = $Color_Stripe_IMG -> colorAllocate (0,0,0);          #Allocate color 'Black';

print "D: Color Stripe image is: '$Col_Stripe_Width x $Col_Stripe_Height'\n";
$Color_Stripe_IMG -> rectangle (1,1, $Col_Stripe_Width -1, $Col_Stripe_Height-1, $Black);
#my $Image = new GD::Image (1000,100);       #HW: For testing Color Stripe...
#my @Color_Stripe;
#Colors run: Full Blue - Partial Blues - Full White - Partial Reds - Full Red
#print "D: Allocate 'Blues': \n";

my @Color_Stripe_Bar;
#Colors run: Full Blue - Partial Blues - Full White - Partial Reds - Full Red
print "D: Allocate 'Blues': \n";
foreach my $C_Color (0..($Config{Color_Strips}/2-1))        #Run: Full Blue to one level
below white
        {
        printf ("%3i ",$C_Color);
        my $Blue_level = 255/($Config{Color_Strips}/2)*$C_Color;   #The (complex)
calculation for the color level
#       print "D: Allocating Color: Blue_level = '$Blue_level'\n";          #works for the
red as well but without the "255-" part
        push @Color_Stripe_Bar, $Color_Stripe_IMG -> colorAllocate
($Blue_level,$Blue_level,255);
        }
print "D: Color_Stripe_Bar: ,|@Color_Stripe_Bar| i.e. has: $#Color_Stripe_Bar +1
divisions\n";
#print "D: $#Color_Stripe, @Color_Stripe\n";                               #Note down
the index of the color just allocated in a 'look-up' table
#print "D: Allocating White: < As mid point >";
push @Color_Stripe_Bar, $Color_Stripe_IMG -> colorAllocate (255,255,255); #The 'White' is
fixed.
print "D: Color_Stripe_Bar: ,|@Color_Stripe_Bar| i.e. has: $#Color_Stripe_Bar +1
divisions\n";
#print "D: $#Color_Stripe, @Color_Stripe\n";
print "\nD: Allocate 'Reds': \n";
foreach my $C_Color (1..($Config{Color_Strips}/2))  #Run: one above 'white' to full red
        {
        printf ("%3i ",$C_Color);
        my $Red_level = 255 - 255/($Config{Color_Strips}/2)*$C_Color;
#       print "D: Red_level = '$Red_level'\n";
        push @Color_Stripe_Bar, $Color_Stripe_IMG -> colorAllocate
(255,$Red_level,$Red_level);
        }
print "\n";
print "D: Color_Stripe_Bar: ,|@Color_Stripe_Bar| i.e. has: $#Color_Stripe_Bar +1
divisions\n";


print "D: Will use color: ";
foreach my $C_color (0..$#Color_Stripe_Bar)
        {
        printf ("%3i ",$C_color);
#       print "D: Drawing box: '$C_color'\n";
        my $X1 = ($C_color) * $Config{Block_Size} *  $Config{Scale};       #Account for off-
center scale: 3,4,5.. to 0,1,2 for plotting
        my $X2 = ($C_color +1)  * $Config{Block_Size} * $Config{Scale};
#       print "D: X1 = '$X1', X2 = '$X2',    ";
#print "D: Will use color = '$Color_Stripe[$C_color]', i.e. A_color: $A_color; C_color:
$C_color;
        printf ("%2i ",$C_color);
        $Color_Stripe_IMG -> filledRectangle ($X1,0,$X2,$Config{Block_Size} *
$Config{Scale},$Color_Stripe_Bar[$C_color]);
        $Color_Stripe_IMG -> rectangle ($X1, 0 , $X2-1, $Config{Block_Size} *
$Config{Scale}-1,$Black);
#       $Color_Stripe_IMG -> stringTTF ($Black, "./fonts/arial.ttf", 20, 0,$X1, 20,
$C_color);
        }
```

Cancel

Figure 15i

```
#Highlight the middle part of the scale:
my $C_color = $#Color_Stripe/2;
my $X1 = $C_color * $Config{Block_Size};     #Account for off-center scale: 3,4,5.. to 0,1,2
for plotting
my $X2 = ($C_color +1)  * $Config{Block_Size};

#$Color_Stripe_IMG -> rectangle ($X1 * $Config{Scale},1,$X2 *
$Config{Scale},$Config{Block_Size} * $Config{Scale}-2,$Black);
#open OUTPUT, ">Color_Stripe.png" or die "Cannot open output file: 'Color_Stripe.png'\n";

#print OUTPUT $Color_Stripe_IMG -> png ();          #Thankfully OO!  The difficult bit!
#close OUTPUT;                           #Will close anyway...
}

#########SUB START
#sub Draw_Classification_Stripe {
#HEY!  This doesn't do anything!!!!
#open OUTPUT, ">Classification_Stripe.png" or die "Cannot open output file:
'Classification_Stripe.png'\n";
#print OUTPUT $Classification_Stripe_IMG -> png ();        #Thankfully OO!   The difficult
bit!
#close OUTPUT;                           #Will close anyway...
#}

#########SUB START
sub Load_Configuration {
#This loads configuration into the main Config hash array.  Defaults are given first:
$Config{Block_Size}                      =       16;     #The size (in Pixels) of each block.
#File names: Hard Wired in version 1_1!
```

Cancel

```
$Config{Clinical_Data_File}              = "./csv/Tabel AML clinical and molecular data
23_07_2003.csv";       #The name of the Clinical Datafile (Comma delimited format).
$Config{Output_File}                     = "485Output.png";                 #Name of the
final generated image.
#Other parameters:
$Config {Block_Lines}                    = "F";  #Whether to draw lines round the (inside) of
the blocks
                                         #NB: Reduces colored area by 1 pixel in both
dimensions
$Config {Draw_Color_Stripe}                      = "T";  #Should a Key be prepared?
$Config {Color_Strips}                   = 40;   #The number of intervening colors in the
'Strip'
$Config {Minimum}                        = -1;   #Assumed minmum of correlation data
$Config {Maximum}                        = +1;   #Assumed minmum of correlation data
$Config {Scale}                                  = 5;    #The multiplication factor for relative
to $Block_Size of the Blocks in the Color Stripe
$Config{Correlation_File}                = "./362
 View  all clustered columnsets .csv";
$Config{Correlation_File}                = "./incoming/485genes.csv";
$Config{Header_Col_1}                    = "FAB";
$Config{Header_Col_2}                    = "WBC";
$Config{Header_Col_3}                    = "FLT3 ITD";
#$Config{Header_Col_4}                   = "FLT3 TKD";
$Config{Header_Col_5}                    = "os";
$Config{Header_Col_6}                    = "efs";
$Config{Header_Col_7}                    = "EVI1";
$Config{Header_Col_8}                    = "CEBP mutant";
$Config{Header_Col_9}                    = "osi";


$Config{Mark_Nulls}                      = "SPOT";

$Config{Mark_Patient_Data}               = "Y";
$Config{Marked_Patient}                          = "XXXXXXXXXXXXXXXXXX";        #Inserts a black
line to demonstrated correspondence / registery between patient CC and classification type.
$Config{Label_Classes}                   = "Y";
```

Figure 15j

```
$Config{Second_Scale_Spacing}        = $Config{Block_Size} * 10;   #The spacing between the
first and the second scale...*10 sets this to ~130% the length of the first scale
$Config{Low_Blood_Count}             = 100; #These were set by MJM because they were "nice
round numbers" they have no scientific justification
$Config{Med_Blood_Count}             = 150; #
$Config{Hi_Blood_Count}                      = 200; #
$Config{Blood_Count_Max}             = 300; #
$Config{EFS_Max}                     = 166;
$Config{OS_Max}                      = 166;
$Config{Graph_Space}                 = 250;
$Config{Font_Size}                   = 15;
#print "D: Reading Configuration Information from STDIN:\n";
#my $Keys_Read=0;
#my @STDIN= <STDIN>;
#if ($STDIN[0] eq "") {return;}
#foreach (@STDIN)
#       {
#       chomp ();
#       unless (/=/)   {die "Error reading cofiguration file: Pattern expected
is:\n'Parameter = Value'\nWhat was found was: '$_'\n";}
#       s/ //g;         #Kill all spaces
#       (my $Key , my $Value) =      split ("=",$_);
#       print "D: Key = '$Key' ; Value = '$Value'\n";
#       $Keys_Read ++;
#       }
#print "D: Finished reading config file: In total '$Keys_Read' extra parameters read\n";
}

#########SUB START
sub Mark_Patient_Data {
#Find number of Colors needed (i.e. find number of catergories:
my $Black = $Image -> colorAllocate (0,0,0);
my $Yellow = $Image -> colorAllocate (255,255,0);   #M6
my $Cyan = $Image -> colorAllocate (0,255,255);              #M5
my $Maroon = $Image -> colorAllocate (176,48,96);   #M4
my $Orange = $Image -> colorAllocate (255,165,0);   #M3
my $Pink = $Image -> colorAllocate (255,105,180);   #M2
my $D_Green = $Image -> colorAllocate (85,107,47);  #M1
my $Green = $Image -> colorAllocate (0,255,0);               #M0
my $Red =      $Image -> colorAllocate (255,0,0);
my $Soft_Green =       $Image -> colorAllocate (128,255,128);
my $Soft_Red =         $Image -> colorAllocate (255,128,128);
my $Low =$Image -> colorAllocate (32,32,32); #12.5% Grey: Low Blood Cell count
my $Med =$Image -> colorAllocate (128,128,128);     #50% Grey: Medium Blood Cell count
my $Hi  =$Image -> colorAllocate (214,214,214);     #87.5% Grey: High Blood Cell count

foreach my $row (0..$#Matrix)              #Cycle through all rows
       {
       my ($x1, $y1, $x2, $y2); #$row; my $Y = $row;
       $x1 = $Config{Block_Size} * $row; $x2 = $x1 + $Config{Block_Size}; #Top left to
Bottom right of a square
       $y1 = $x1; $y2 = $y1 + $Config{Block_Size}-1;
              #This is the diagonals of the square....
       my $x_cent = int ( ($x2 - $x1 ) /2) + $x1; my $y_cent = int ( ($y2 - $y1 ) /2)  +
$y1;   #The center might be useful...calculation is over complex, but hey - it's standard!
       my $C_Class = $Classification_1{$Patient_ID{$row}};        #Just a convenience
really....
       print "D: Classification of Patient ($Patient_ID{$row}) #'$row' = '$C_Class'\n";
       $Image -> filledRectangle ($x1, $y1, $x2, $y2, $White);           #Blank blocks on
diagongal
#print "D: $#Color_Stripe, @Color_Stripe\n";                              #Note down
the index of the color just allocated in a 'look-up' table
#print "D: Allocating White: < As mid point >";


#Ok!  This is where the logic begins...
#Do classification #1: FAB Type:
       if ($C_Class =~ m/Mx/)
              {      #Ie.  A mixed system...
              #Draw Spot....
              print "D: Mixed classification found - drawing spot\n";
```

Cancel

Figure 15k

```
#               $Image -> line ($x1,$y1,$x2,$y2,$Black);

                $Image -> arc ($x_cent,$y_cent,$Config{Block_Size}, $Config{Block_Size}, 0
,360 , $Black);
                $Image -> fill ($x_cent,$y_cent, $Black);
                print "D: Diagonal block runs: $x1, $y1 through center at $x_cent, $y_cent
to: $x2, $y2\n";
                }
        if ($C_Class eq "")
                {       #Ie.  Missing Classification...
                print "D: Missing Classification: Drawing a cross\n";
                $Image -> line ($x1, $y1, $x2, $y2, $Black);
                $Image -> line ($x2, $y1, $x1, $y2, $Black);
#               next;           #Easy eh?  (;-)
                }

        if ($C_Class =~ m/M/ and not $C_Class =~ m/Mx/)
                {
                my $Block_color;
                my $Catergory = substr ($C_Class, 1,1);
                print "D: Catergory = '$Catergory'\n";
#               $Block_color = $Cat_bottom_color + $Catergory;
                if ($Catergory == 6)   {$Block_color = $Yellow;}
                if ($Catergory == 5)   {$Block_color = $Cyan;}
                if ($Catergory == 4)   {$Block_color = $Maroon;}
                if ($Catergory == 3)   {$Block_color = $Orange;}
                if ($Catergory == 2)   {$Block_color = $Pink;}
                if ($Catergory == 1)   {$Block_color = $D_Green;}
                if ($Catergory == 0)   {$Block_color = $Green; }
                print "D: Will use color: '$Block_color'\n";
                $x2 = $x1 + 20*$Catergory+$Config{Block_Size} -1;
                $Image -> filledRectangle ($x1, $y1, $x2 , $y2, $Block_color);
                if ($Config{Label_Classes}    eq      "Y")
                        {
                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf", 15, 0, $x2+10,
$y2, $Catergory);
                        }
                }
        $Patient_IDs -> stringTTF ($Black, "./fonts/Courier.ttf", $Config{Font_Size}, 0, 1,
$y2,$Patient_ID{$row} );
        if ($Patient_ID{$row} eq $Config{Marked_Patient})          #This is used to check
the 'register' between the CC data and the Patient Classification.
                {
#               my $Block_color = $Black;
                my $Catergory = substr ($C_Class, 1,1);
                print "D: Marking Patient: '$Patient_ID{$row} ' using color: BLACK\n";
                my $Catergory =        10;
                $Image -> filledRectangle ($x1, $y1, $x2 + 20 * $Catergory, $y2, $Black);
                }

#Now something similar for classification #2 (Blood Cell Count):
        $x1=$x1 + $Config{Graph_Space};       #ie. give some space between the two scales
        $x2 =  $x1 + $Config{Block_Size};
        my $Blood_Count = $Classification_2{$Patient_ID{$row}};
        print "D: Blood count = '$Blood_Count'\n";
        if ($Blood_Count == undef)
                {
                print "D: Missing Blood Count Classification: Drawing a cross\n";
                $Image -> line ($x1, $y1, $x2, $y2, $Black);
                $Image -> line ($x2, $y1, $x1, $y2, $Black);
                }
                else
                {
                my $Bar_Length = $Blood_Count / $Config{Blood_Count_Max} * 200;
                Draw_blood_bar ($Med, $Blood_Count,$x1, $y1,  $Bar_Length);
                }
        #$Config{Blood_Count_Max}


#Now something similar for classification #3 (FLT ITD):
        $x1=$x1 + $Config{Graph_Space};      #ie. give some space between the two scales
```

Cancel

Figure 151

```
        my $FLT_Class = $Classification_3{$Patient_ID{$row}};
        print "D: FLT3 Class = '$FLT_Class' for Patient: '$Patient_ID{$row}'\n";
        if ($FLT_Class eq "")
                {
                print "D: Missing FTL Classification: Drawing a cross\n";
                $x2 =  $x1 + $Config{Block_Size};
                $Image -> line ($x1, $y1, $x2,  $y2, $Black);
                $Image -> line ($x2, $y1, $x1,  $y2, $Black);
                }
                else
                {
                if  ($FLT_Class =~ m/Pos/i or $FLT_Class =~ m/Yes/i)
                        {
                        $x2=$x1 + 150;
                        $Image -> filledRectangle ($x1, $y1, $x2, $y2, $Soft_Red);
                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf",
$Config{Font_Size}, 0, $x2+10, $y2-2, "Pos");
                        }
                        else
                        {
                        $x2=$x1 + 75;
                        $Image -> filledRectangle ($x1, $y1, $x2, $y2, $Soft_Green);

                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf",
$Config{Font_Size}, 0, $x2+10, $y2-3, "Neg");
                        }
                }



#Now something similar for classification #5  (OS):
        $x1=$x1 + $Config{Graph_Space};       #ie. give some space between the two scales
        $x2 =  $x1 + $Config{Block_Size};
        my $OS = $Classification_5{$Patient_ID{$row}};
        print "D: OS = '$OS'\n";
        if ($OS eq "")
                {
                print "D: Missing OS Classification: Drawing a cross\n";
                $Image -> line ($x1, $y1, $x2,  $y2, $Black);
                $Image -> line ($x2, $y1, $x1,  $y2, $Black);
                }
                else
                {
                my $Bar_Length = $OS / $Config{OS_Max} * 200;
                Draw_blood_bar ($Med, $OS,$x1,  $y1, $Bar_Length);
                }
        #$Config{Blood_Count_Max}

#Now something similar for classification #6  (EFS):
        $x1=$x1 + $Config{Graph_Space};       #ie. give some space between the two scales
        $x2 =  $x1 + $Config{Block_Size};
        my $EFS = $Classification_6{$Patient_ID{$row}};
        print "D: $Patient_ID{$row} EFS = '$EFS'\n";
        if ($EFS eq "")
                {
                print "D: Missing EFS Classification: Drawing a cross\n";
                $Image -> line ($x1, $y1, $x2,  $y2, $Black);
                $Image -> line ($x2, $y1, $x1,  $y2, $Black);
                }
                else
                {
                print "D: Testing Dead/ alive status:
'",$Classification_9{$Patient_ID{$row}},"'\n";
                my $Bar_Length = $EFS / $Config{EFS_Max} * 200;
                if ($Classification_9{$Patient_ID{$row}} eq "alive")
                        {Draw_blood_bar ($Soft_Green, $EFS,$x1, $y1, $Bar_Length);}
                        else
                        {Draw_blood_bar ($Soft_Red, $EFS,$x1, $y1, $Bar_Length);}
                }
```

Canc

Figure 15m

```
#Now something similar for classification #7 (EVI1):
        $x1=$x1 + $Config{Graph_Space};       #ie. give some space between the two scales

        my $EVI1_Class = $Classification_7{$Patient_ID{$row}};
        print "D: EVI1 Class = '$EVI1_Class' for Patient: '$Patient_ID{$row}'\n";
        if ($EVI1_Class eq "")
                {
                print "D: Missing EVI1 Classification: Drawing a cross\n";
                $x2 =  $x1 + $Config{Block_Size};
                $Image -> line ($x1, $y1, $x2, $y2, $Black);
                $Image -> line ($x2, $y1, $x1, $y2, $Black);
                }
                else
                {
                if ($EVI1_Class =~ m/Pos/i or $EVI1_Class =~ m/Yes/i)
                        {
                        $x2=$x1 + 150;
                        $Image -> filledRectangle ($x1, $y1, $x2, $y2, $Soft_Red);
                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf",
$Config{Font_Size}, 0, $x2+10, $y2-2, "Pos");
                        }
                        else
                        {
                        $x2=$x1 + 75;
                        $Image -> filledRectangle ($x1, $y1, $x2, $y2, $Soft_Green);

                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf",
$Config{Font_Size}, 0, $x2+10, $y2-3, "Neg");
                        }
                }
#CEBP mutant to go in!
#Now something similar for classification #8 (CEBP):
        $x1=$x1 + $Config{Graph_Space};       #ie. give some space between the two scales

        my $CEBP_Class = $Classification_8{$Patient_ID{$row}};
        print "D: CEBP Class = '$CEBP_Class' for Patient: '$Patient_ID{$row}'\n";
        if ($CEBP_Class eq "")
                {
                print "D: Missing CEBP Classification: Drawing a cross\n";
                $x2 =  $x1 + $Config{Block_Size};
                $Image -> line ($x1, $y1, $x2, $y2, $Black);
                $Image -> line ($x2, $y1, $x1, $y2, $Black);
                }
                else
                {
                if ($CEBP_Class =~ m/Pos/i or $CEBP_Class =~ m/Yes/i)
                        {
                        $x2=$x1 + 150;
                        $Image -> filledRectangle ($x1, $y1, $x2, $y2, $Soft_Red);
                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf",
$Config{Font_Size}, 0, $x2+10, $y2-2, "Pos");
                        }
                        else
                        {
                        $x2=$x1 + 75;
                        $Image -> filledRectangle ($x1, $y1, $x2, $y2, $Soft_Green);

                        $Image -> stringTTF ($Black, "./fonts/Courier.ttf",
$Config{Font_Size}, 0, $x2+10, $y2-3, "Neg");
                        }
                }

        next;
        }
#return ($Cat_bottom_color, $Number_of_colors);
}

sub Draw_blood_bar {
(my $color, my $Count, my $x, my $y, my $Length)    =     @_;
$Image -> filledRectangle ($x, $y, $x + [illegible] Size}-1, $color);
```

[illegible]

Figure 15n

```
$Image -> stringTTF (1, "./fonts/Courier.ttf", $Config{Font_Size}, 0, $x + $Length + 10, $y
+ $Config{Block_Size}-1, int ($Count));

}

################START SUB
#sub Draw_Classification_Stripe {
#Er?  Finishing this would be a good idea....
#Hey!  This doesn't do anything!
#for my $C_Class (1..$Classes)
#       {


#       }
#}


sub Label_Class {
(my $x, my $y, my $Cat) = @_;
print "D: LABEL_CLASS: Got the data: [X,Y,Cat] '$x', '$y', '$Cat' passed\n" ;

}


sub Top_Color_Print {

print "D:       [Allocating new color of index: '$Top_Color']\n";
$Top_Color ++;
}

sub Allocate_Catergory_range {
my %Classes;
my $Number_of_Classes=0;
foreach my $C_Patient (keys %Classification_1)          #Cycle through all
classifications
        {
#       print "D: Classification of Patient: '$C_Patient' =
'$Classification_1{$C_Patient}'\n";
        unless (exists $Classes{$Classification_1{$C_Patient}})     #Check whether this
classification has been seen before.
                {                                       #Ok, it hasn't:
                print "D: Found new Class: '$Classification_1{$C_Patient}'\n";
                $Classes{$Classification_1{$C_Patient}} = $Classification_1{$C_Patient};
                #Add it to the Hash Array
                $Number_of_Classes ++;                  #Add 1 to the tally of classes
                }
        }
print "D: Number of FAB Classes (patient catergories)  = '$Number_of_Classes'\n";#Useful to
know
print "D:       Allocate 'Catergory Colors': \n";
my $CC_max_color = $#Color_Stripe;
my $Cat_bottom_color = $CC_max_color + 3;
print "D: Last Color Allocated for CC Matrix: $CC_max_color '$Cat_bottom_color'\n";
my $Number_of_colors = $Number_of_Classes - 3;
foreach my $C_Color (0..$Number_of_colors)  #Ie, pickup where the CC data left off
        {
        printf ("%3i ",$C_Color);
        my $Red_level = int (255 / $Number_of_colors * $C_Color);  #The (complex)
calculation for the color level
        print "D: For $C_Color: Red_level (needed to alter Green to Yellow) = '$Red_level',
i.e. Color:",($C_Color+$Cat_bottom_color),"\n";          #works for the red as well but
without the "255-" part
#       push @Color_Stripe,
$Image -> colorAllocate ($Red_level,255,0);
        }
my $Cat_top_color = $#Color_Stripe;          #Don't think this is actually used...nice to
know though!
print "D: Catergory colors will range from: $Cat_bottom_color to '",$Cat_bottom_color +
$Number_of_colors,"'\n";
}
```

Cancel

Figure 15o